Enhanced Performance Analysis of Multi-core Applications with an Integrated Tool-chain

Using Scalasca and Vampir to Optimise the Metal Forming Simulation FE Software INDEED

Thomas WILLIAM [a], Hartmut MIX [b] Bernd MOHR [c], René MENZEL [d]
Felix VOIGTLÄNDER [c]

[a] *GWT-TUD GmbH Dresden, Germany*
[b] *Technische Universität Dresden, ZIH, Germany*
[c] *Jülich Supercomputing Centre, Germany*
[d] *GNS mbH Braunschweig, Germany*

Abstract. Programming and optimising large parallel applications for multi-core systems is an ambitious and time consuming challenge. Therefore, a number of software tools have been developed in the past to assist the programmer in optimising their codes. Scalasca and Vampir are two of these performance-analysis tools that are already well established and recognised in the community. While Scalasca locates, quantifies, and presents performance flaws (defined as event patterns) in a compact and simple to use tree representation, the Vampir framework visualises all details of the function call sequences and communication patterns in a great variety of timeline or statistic displays. Scalasca locates and presents the problems found and their severity but gives limited practical hints on the case history to find a solution. Looking at the huge amount of detailed data in Vampir can make it hard to find the interesting spots in spatial and temporal resolution for identification of problems and selection for further investigation. This observation led to the combination of these tools resulting in a more productive tool-chain. In this paper we present our approach to connect Scalasca and Vampir through a common bus system. A practical real-world example is provided and the beneficial impact of the integration on the work-flow is shown by optimising the adaptive mesh refinement and load balancing phases of the metal forming simulation FE software INDEED.

Keywords. MPI, OpenMP, performance optimisation, tools integration, Scalasca, Vampir

Introduction

Performance analysis and optimisation of applications are important phases of the development cycle. Like testing and debugging it should be imperative. It is an important precondition to guarantee efficient usage of expensive and limited computing resources.This means obtaining the results with minimum resource usage and cost. Furthermore, it is important for scalability, i.e. being able to achieve the next larger simulation with the

given resources. For that, it is particularly important to exploit as much of the theoretically available performance as possible.

The task of the analysis phase is to measure the actual performance on a given platform in terms of computing speed or throughput as well as resource consumption in regard to run-time or memory requirement or storage space. Secondly, performance analysis has to identify opportunities for performance improvement or reduction of resource usage. Tracing requires the modification of the target application in order to detect event occurrences, this step is also known as instrumentation. The events gathered during tracing are simply points of interest in the course of program execution. The most common event types are entering and leaving a program region, sending or receiving messages, collective operations, and performance counter samples which provide a scalar value at a point in time. The tracing records all individual events along with general properties like exact time stamp and the originating process or thread as well as further event type specific properties. Thereby, tracing allows to investigate single events. Information about events permit to infer about application flow like function calls (or general basic blocks), communication or other activities relative to individual processes or threads. This enables trace-based tools to identify variations in the dynamic behaviour of the program over many iterations.

In this paper we will show the benefit that can be gained by combining two tracing tools which provide different but complementary functionality. The paper is structured in the following way: At first we introduce the two tool-chains Scalasca and Vampir and show how we integrated them. Then the application INDEED which we analysed to demonstrate our new integrated tool-set is described. This is followed by the optimisations we achieved. Finally we refer to related work and draw a conclusion.

Scalasca / KOJAK

Scalasca is an automatic performance evaluation system for C/C++ or Fortran parallel applications. It is developed by Forschungszentrum Jülich, Germany. The Scalasca framework generates event traces from running applications and automatically searches them off-line for execution patterns indicating inefficient application behaviour. These patterns are used during the analysis process to recognise, classify, and quantify the inefficiencies in the application. The performance problems recognised by Scalasca include inefficient use of the parallel programming models MPI, OpenMP, and SHMEM as well as low CPU and memory performance. The analysis process automatically transforms the traces into a compact call-path profile that includes the execution time penalties caused by the different patterns broken down by call path and process or thread. The predecessor tool, called KOJAK, searched the patterns sequentially. The new Scalasca tool performs the search in parallel which allows to efficiently analyse parallel programs running on many thousand cores (see figure 1, lower part). For a detailed description of KOJAK including a complete list of all patterns see [4,5]. Scalasca is described in detail in [6].

The call-path profile produced by the analysis can be viewed using the CUBE presenter. CUBE is a generic tool for displaying a multidimensional performance space consisting of the dimensions (i) metric, (ii) call path, and (iii) system resource. Each dimension is represented as a tree browser that can be collapsed or expanded to achieve the desired level of granularity or specialisation. The tree browsers are coupled such that

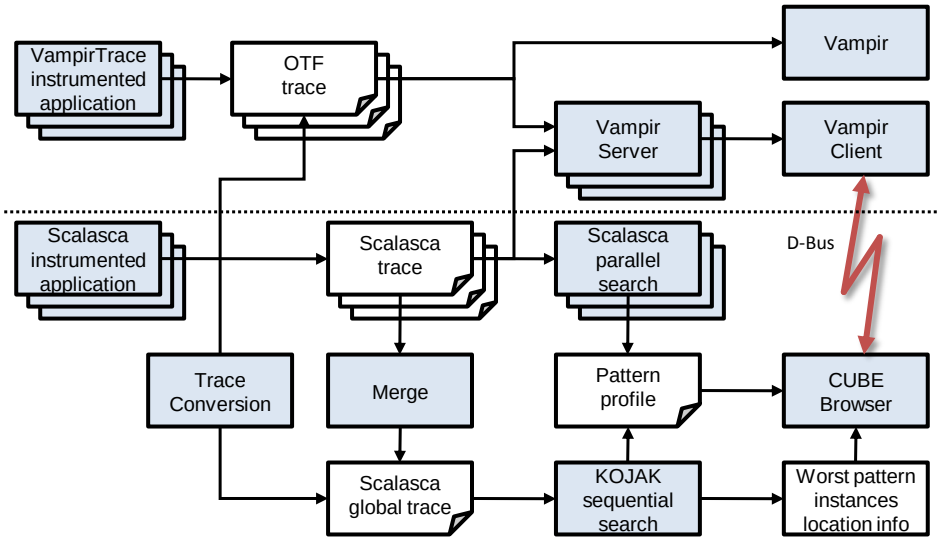

Figure 1. Combined workflow of the Vampir (top) and Scalasca (bottom) toolsets as of Fall 2009. Stacked boxes indicate parallel execution or multiple files. Work is under way to extend the Scalasca parallel search to also produce the "worst pattern instances location information" (replacing the inefficient old merge and sequential search facilities) and to design and implement a common Vampir/Scalasca measurement system and trace format, both simplifying the future work-flow tremendously.

the penalty caused by a particular performance problem can be identified for each call path and process or thread (see figure 3).

Vampir

The Vampir tool family [10] consists of the instrumentation and measurement component VampirTrace [11] and the visualisation applications Vampir and VampirServer [9]. It is developed by the Technische Universität Dresden for analysing the runtime behaviour of parallel MPI/OpenMP programs and visualises the program execution by means of event traces (see figure 1, upper part).
The visualisation takes place after the monitored program has been completed, using traces recorded during its execution. VampirServer introduces parallel performance data evaluation concepts which are implemented in a client-server framework. The server component can be executed on a segment of the parallel production environment. The corresponding client can run on a remote desktop computer to visualise the performance results graphically. This has the advantages that performance data which tends to be bulky is kept where it was created, that parallel processing significantly increases the scalability of the analysis process, and that very large trace files can be browsed and visualised interactively. The visualisation client translates the condensed performance data into a variety of graphical representations providing developers with a good understanding of performance issues concerning their applications. This allows for quick focusing on appropriate levels of detail which provide the detection and explanation of various performance bottlenecks such as load imbalances and communication deficiencies.

Integration of Scalasca and Vampir

Scalasca scans execution traces for event patterns representing inefficiencies and the patterns found are categorised and ranked. The analysis results can be visualised using the CUBE browser. The users can identify the most severe problems in their code, locate the place in the code where the problem occurs and can see how its severity is distributed across the threads and processes. To find a solution for these problems this information is sometimes not sufficient. In some cases it is necessary to know the temporal context that led to the problem. This can be summarised as knowing the problem but having no real hint on its cause.

Using the Vampir tool chain a full trace of the communication patterns and call sequences can be visualised. There are many different views on the data which allow a very detailed analysis of the complete execution history but the vast amount of data complicates the search for bottlenecks. So in terms of finding performance flaws this can be cumbersome like ”finding the needle in the haystack”.

Integrating both tools enables the user to combine the advantages of Scalasca with those of Vampir (see figure 1). The Scalasca pattern search has been extended to also remember the location of the worst instance of each recognised performance problem for every call-path effected by the problem. The CUBE browser now features a new option in the file-menu to connect to Vampir. If this is selected, starting the Vampir client and server as well as connecting the two, and the loading of the necessary trace-file are all handled automatically. This is more user-friendly because the latest Vampir version can now also load and interpret Scalasca traces directly. CUBE then opens the global timeline as a default view in Vampir. Right clicking a pattern in CUBE reveals a new option ”Max severity in trace browser” with which Vampir is instructed to zoom into that portion of the trace where the worst instance of this pattern occurred. Now the user can leverage all the possibilities available in Vampir to explore the trace to further investigate the history which led to the problem located.

Figure 2. D-Bus interface

The remote control of Vampir through the CUBE browser or any other program is realised with the D-Bus protocol (see [12]). Using the D-Bus interface exposed by Vampir

is fairly simple. The developer can implement it using the mechanism provided by Qt, C or C++. Figure 2 shows the interface of Vampir as it can be seen on the D-Bus using introspection.

INDEED

INDEED (INnovative DEEp Drawing) is an implicit finite element software for the numerical simulation of sheet metal forming processes like deep drawing, roll forming, crash forming, hydro forming of tubes and welded blanks and hydro mechanical deep drawing [1]. The main characteristics of INDEED are summarised as follows :

- an implicit integration scheme based on a total Lagrangian formulation with contact and equilibrium iterations fulfilled at the end of each increment
- specially developed thick shell elements based on the so-called director shell theory [2] with 21 degrees of freedom per element
- fast contact-search algorithm based on an "active-set-strategy" and taking locally smoothed second-order contact-surfaces into account
- adaptive mesh refinement accounting for tool radii and local blank curvature

The requirements for metal forming simulation software are increasing steadily: on one hand, more time consuming sophisticated material models, element and contact formulations are necessary to obtain accurate results of complex phenomena like spring-back prediction or surface defects detection, while on the other hand a cost-effective manufacturing process with a short development time is the main focus. To meet these requirements, a parallelization of the software was inevitable. At first, the most time consuming routines were parallelized with OpenMP directives, but currently a domain decomposition approach based on the message-passing interface (MPI) is also under development to benefit from massively parallel cluster technology. Special care must be taken in this case in an adaptive mesh refinement step. To allow for a correct mesh connectivity, some information has to be exchanged across sub-domain boundaries. Furthermore, the independent mesh refinement of each sub-area might result in an imbalance of the computational load of the processors in subsequent steps. That problem is solved in a second phase via a revaluation of the domains. For this purpose the PARMETIS[3] library can be used, but also own coordinate-based repartitioning algorithms were developed. The following section gives an example for the analysis and optimisation of both phases with the approach described in this paper.

Optimisation

INDEED uses an adaptive mesh refinement accounting for the tool radii and the local blank curvature. The recently developed parallel version based on domain decomposition induced new challenges during the rezoning step. The algorithm has to safeguard the correct element connectivity across adjacent sub-domains while rezoning. An update of the information concerning the neighbouring sub-domains has to be performed. Additionally the repartitioning of the blank has to be taken into account as the computational load of the domains becomes too unbalanced. With these background information it is now possible

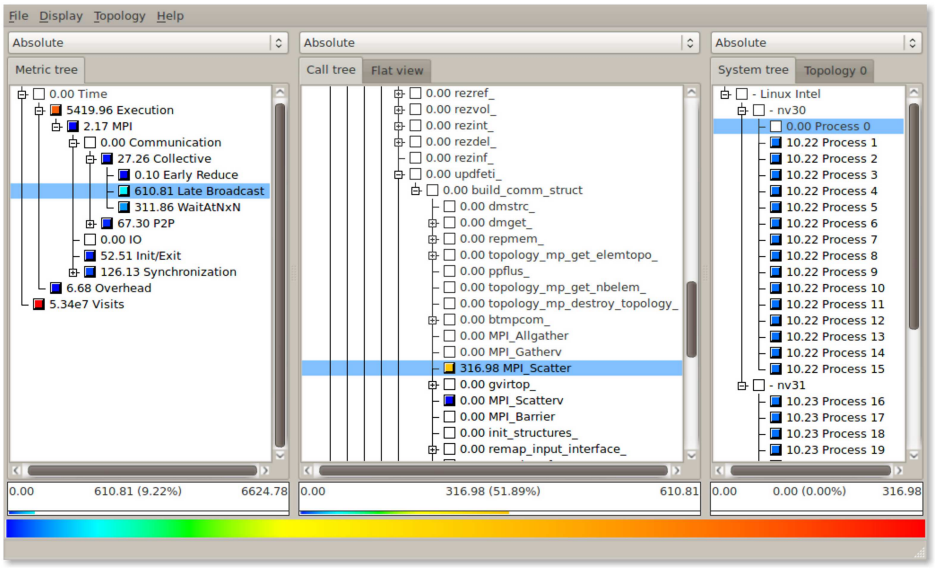

Figure 3. CUBE view showing the severity of the *Late Broadcast* pattern located at the MPI_SCATTER call in the build_comm_struct function

to investigate the performance problems located by Scalasca in the CUBE browser (see figure 3). Most of the time is lost in the MPI part of the application due to the pattern called *Late Broadcast*. Stepping down into the call-tree reveals an inefficient `MPI_Scatter` in the function `build_comm_struct` to be the main reason for this problem. The system tree on the right shows that process 0 seems to be the root-node of the broadcast operation providing the information too late. This knowledge alone is not enough as we have no indication of why process 0 took so long to enter `MPI_Scatter`. Using our new approach described above it is possible to investigate the history of this problem directly in Vampir (see figure 4). The figure reveals that process 0 is still computing while all the other processes are already waiting for data in the `MPI_Scatter`. The function only called by process 0 is `find_neighbour_coor` (see figure 4) and is responsible for the update of neighbour relationships during the rezoning or repartitioning. Although this function only takes a small amount of time to finish it has to be called for all edge-nodes of each zone or partition. Therefore the coordinates of all edge nodes are collected and send from each process to the process 0 were the recalculation takes place. As can be seen in the Vampir timeline display, this behaviour leads to the *Late Broadcast* pattern. The solution in this particular case is fairly easy as this type of coordinate matching can be done in parallel. Implementing this (and a few other optimizations) and rerunning the application results in a decrease of time lost in the *Late Broadcast* pattern from 610 seconds down to 28 (see figure 5). A positive side effect is that also the synchronisation time dropped down from 126 to 16 seconds as well as the *WaitAtNxN* pattern (311 down to 233) while the peer-to-peer communication time slightly increased from 67 to 74. So, by parallelizing this single function and thereby restructuring the load-balancing and rezoning steps an overall benefit to the runtime could be accomplished.

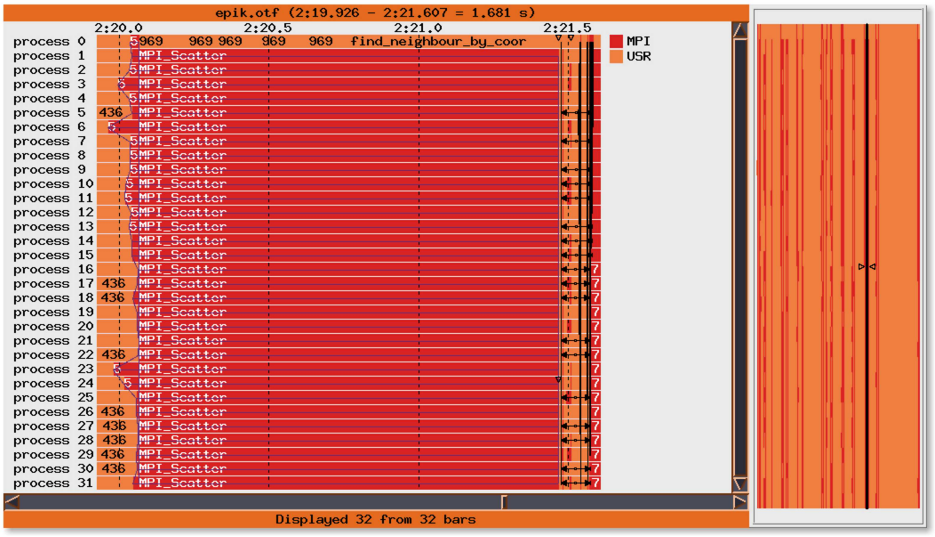

Figure 4. Same *Late Broadcast* Pattern shown in global timeline of Vampir

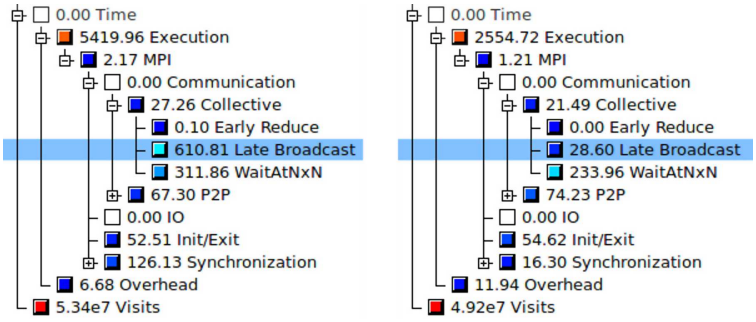

Figure 5. Different severity values before and after the optimization step

Related Work

Parallel performance tools so far were typically only very loosely coupled, for example by providing performance data (profiles, traces) converters. While this allows the use of another tool to analyse the same data in a different way, it was left to the users to find the corresponding data or displays in the second tool for a problem they found in the first tool. Also, it is often the case that information gets lost in the conversion. In our approach, the tools are very tightly coupled via a modern inter-process communication mechanism.

The latest version of the Paraver visualization tool [7] provides a remote control interface [8] that allows coupling Paraver with other tools comparable to what we did. However, the inter-process communication is based on the (unreliable and unportable) UNIX signal mechanism, and is not yet documented and published, which makes a more detailed comparison impossible.

Conclusions

In this paper we described the integration of the well-established parallel performance tools Vampir and Scalasca into a coherent integrated tool-set. Scalasca allows to quickly locate and rank parallel programming model related performance problems in application codes. Vampir provides very powerful visualisation and analysis functions to investigate the execution history which led to these problems and therefore find a potential solution easily. The newly developed integration allows to automatically start Vampir with the right trace data from within Scalasca and to display the execution history of problems located by Scalasca in timeline displays of Vampir with one mouse click streamlining the whole analysis process. We demonstrated the usefulness and effectiveness of these new features by describing how we used these tools for the optimisation of the adaptive mesh refinement and load balancing phases of the industrial metal forming simulation finite-element software INDEED. While the example used in this paper only used MPI, our software supports the same features also for hybrid OpenMP/MPI applications.

Acknowledgments

This work has partially been supported by the German Ministry for Research and Education (BMBF) through the ITEA2 project "ParMA"(No 06015, June 2007 – May 2010).

References

[1] Mathematical Modelling of the Sheet Metal Forming Process with INDEED. El Rifai Kassem, K. et al. (J.-L. Chenot, R. D. Wood. ed.). Numerical Methods in Industrial Forming Processes. NUMIFORM 92. Balkema. 1992

[2] Schalenelement mit 6 kinematischen Freiheitsgraden bei großen Verschiebungen. H. Schoop. ZAMM 67. 4 : 237-239, 1987

[3] PARMETIS Parallel graph partitioning and sparse matrix ordering library Version 3.1. George Karypis and Vipin Kumar, University of Minnesota, Department of Computer Science and engineering, 2003

[4] F. Wolf, B. Mohr: Automatic performance analysis of hybrid MPI/OpenMP applications, *Journal of Systems Architecture*, Vol. 49, No. 10-11, 421–439, 2003.

[5] F. Wolf, B. Mohr, J. Dongarra, S. Moore: Automatic analysis of inefficiency patterns in parallel applications, *Concurrency and Computation: Practice and Experience*, Vol. 19(11), 1481–1496, 2007.

[6] M. Geimer, F. Wolf, B. J. N. Wylie, B. Mohr: A scalable tool architecture for diagnosing wait states in massively parallel applications, *Parallel Computing*, Vol. 35, 375–388, 2009.

[7] J. Labarta, J. Giménez, E. Martinez, P. González, H. Servat, G. Llort, X. Aguilar: Scalability of visualization and tracing tools, in: Proceedings of Parallel Computing (ParCo), Málaga, Spain, 2005.

[8] Private communication with Paraver developer, 2009.

[9] M. Müller, A. Knüpfer, M. Jurenz, M. Lieber, H. Brunst, H. Mix, W. E. Nagel: Developing Scalable Applications with Vampir, VampirServer and VampirTrace *In C. Bischof, M. Bücker, P. Gibbon, G. Joubert, T. Lippert, B. Mohr, F. Peters: "Parallel Computing: Architectures, Algorithms and Applications"*, Vol. 15, pp. 637-644, ISBN: 978-1-58603-796-3, IOS Press, 2007

[10] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, W. E. Nagel: The Vampir Performance Analysis Tool-Set, *In Tools for High Performance Computing, Proceedings of the 2nd International Workshop on Parallel Tools*, Pages 139-155, Springer, 2008

[11] A. Knüpfer, R. Brendel, H. Brunst, H. Mix, W. E. Nagel: Introducing the Open Trace Format (OTF) *In Vassil N. Alexandrov, Geert Dick van Albada, Peter M. A. Sloot, Jack Dongarra (Eds): Computational Science - ICCS 2006: 6th International Conference, Reading*, UK, May 28-31, 2006, Proceedings, Part II, Springer Verlag, ISBN: 3-540-34381-4, pages 526-533, Vol. 3992, 2006

[12] D-Bus Specification and Documentation *http://www.freedesktop.org/wiki/Software/dbus*